POSIX SHELL

Team : Round Robin

Introduction

Portable Operating System Interface for uni-X (POSIX) is a set of standards given by the IEEE and issued by ANSI and ISO. The aim is to simplify the task of cross-platform software development by setting up a set of guidelines for operating system vendors to follow. It specifies the essential features of a fully POSIX-compliant operating system. Ideally, a developer should have to write a program only once to run on all POSIX-compliant systems.

The POSIX specifications describe an operating system that is like Unix. In Linux, a shell offers an interface for a Unix system to execute commands more easily. A shell collects an input from a user and executes a program according to that input. We can use a shell to perform various operations. In this project, we have developed a working POSIX compatible shell with a subset of feature support of our default shell.

Features Implemented

I) Autocomplete on pressing Tab

II) Basic shell commands such as grep, pwd, cd, cat, head, tail, chmod, exit, history, clear etc.

III) I/O redirection with ‘<’, ‘>>’ and ‘>’ for one source and one destination. Example: cat abc.cpp > def.txt

IV) Generic piping support for any number of pipes. Example: cat xyz.txt | head -5 | tail -2 | sort

V) History: All the valid commands entered by the user are stored

VI) Piping with I/O Redirection. Example: cat hello.cpp | head -10 | tail -4 | sort > hi.txt

VII) Supported the initialization variables like HOME, PATH
USER, HOSTNAME, PS1

VIII) A configuration file, .myrc is maintained. Our program reads this on startup and sets the environment accordingly.
This file contains alias and default applications that we use to open any file.

IX) Background: ‘&’ can be passed as the last token of the current command for background command execution. Example: ls -l &, sleep 50 &

X) Foreground: Brings a process from the background to the foreground

XI) Alarm. This command reminds us of the msg after t seconds. Example: alarm t msg

XII) Export: The export command exports a variable to the environment of child processes. Example: export val=5. This command will export val to another child process

XIII) Prompt look via PS1 is handled

XIV) Association of “~” with the HOME variable

Approach

I) As a part of preprocessing, we created trie for autocompletion and history search

II) We created .myrc file for handling alias case

III) A command is fetched by processing each character input in the shell.

IV) Then the command is passed to splitOnDelimiter() function which splits the command wherever it finds a ‘;’.

V) Then, we parse the command and execute it. we create different versions parallelly for:

   a. Pipe handling
   b. Execute history etc.

VI) In execute function, we are handling different Linux commands.

VII) If the path is not specified, the commands are searched from directories specified in $PATH

VIII) For history command, wen use history.txt file which permanently saves all the valid passed commands. The number of commands in history is same as the values stored in environment variable

IX) For alias command, we use .myrc file to store all the alias commands

X) For autocompletion, we use the trie data structure which stores all the commands and all the file paths

XI) Multiple commands separated by ‘|’ can be executed where the output of the command preceding a command WILL BE TAKEN AS INPUT for the next command using OS pipes.

XII) For alarm, each alarm is implemented using a separate thread and the alarms are session independent

Work Distribution

Soumadeep Acharya	**Shell(), auto complete, history, alias, fetch command, terminal Position setting, check I/O, Prompt(), program structure design, environment variable.**
Subhadeep Biswas	**Arrow Functionalities, tab completion, executeCommand, I/O redirection, command parsing, set/unset and resolving Environment variables, Program structure design.**
Sumonto Chatterjee	**PipeHandler(), Alarm, Export, Foreground, Background, Exit(), sense character from input and processing command, splitOnDelimiter(), Program structure design.**
Venkata Sriram D	**I/O redirection, auto complete, fetch command, input processing, Program structure design. Designing report.**

Problems Faced & Solutions

1. Alarm command implementation: -
   We tried to implement alarm by running sleep() in a child process but, we faced problems related to conversion between raw mode and canonical mode. We fixed it by implementing alarm using threading
2. Pipe implementation: -
   We faced difficulty while handling multiple pipes. We processed the pipes serially. Unwanted spaces were handled.
3. Alias command implementation: -
   We faced a lot of problems while integrating separately implemented code for alias.
4. File Handling: -
   We faced issues while handling the cases when the file was non-existent.
5. Invalid paths were also handled.
6. Non-existent commands will not crash the program.

Learnings

- We learnt about the flow of execution of various commands separated by ‘;’, ‘|’, etc. And how I/O redirection works.
- Learned about differences between canonical and raw mode.
- Input processing.
- Command manipulation using c default libraries.
- Data structure trie and its usage in autocompletion and search.
- Terminal escape sequences.
- Using escape sequences for controlling behavior of terminal.
- Background and foreground processing.

- Child process creation using fork and exec.
- Thread creation using pthread.
- execv.
- Pipes & file descriptors.

Results

- A fully functional shell with restricted command set is implemented with some additional functions such as alarm.

Conclusion

POSIX specification gives us a lot of flexibility as to which systems can run our program. For instance, when pipes are not used then for the same task, we would have to write a lot of code. Creating a new child simplifies the process, using pipes and forking. The use of I/O redirection is one more example. Without it, our speed is a bottleneck for the execution of programs. But I/O redirection ensures the automation of input. It is also useful when we debug files.